

《代码管理: git 使用规范》

作者: 王琦

时间: 2016 年 4 月 24 日

最后修改: 2017 年 5 月 2 日

摘要: 本文先给出了***代码管理工具***的下载地址, 再描述了项目的***创建方式***, 之后以***分支管理***、***代码审查*** (自我审查与他人审查) 为例, 指出了在开发过程中应遵循的行为规范和实现方式。希望大家在开发的过程中, 能遵守规范, 掌握在*命令行*或图形界面下进行代码管理的方法。

1. 工具下载

下载[Github desktop](https://desktop.github.com/)

下载[Sourcetree](https://www.sourcetreeapp.com/)

2. 项目创建

2.1 配置公钥

配置公钥的目的, 是使得今后在进行git操作时, 不用**每次都**需要进行身份验证。

配置教程可参考[配置SSH公钥](https://coding.net/help/doc/git/ssh-key.html)

2.2 克隆项目

```
git clone git@your_repo_name/your_project_name.git
```


3. 项目开发

3.1 分支管理

在此我们先给出分支管理的*逻辑*, 在本节末给出完整的git指令

3.1.1 **分支创建**

每开发一个功能:

- 都应该基于当前最新的dev分支, 去创建一个新的分支
- 该分支名应当*自解释*了当前分支负责开发的功能——给分支取一个有意义的名字**很重要**

假如你正在dev分支下, 你应该使用

```
git checkout -b your_new_branch
```

去创建一个, 基于dev分支的, 名为your_new_branch的新分支。

3.1.2 将发生修改的文件添加到commit list

在完成当前分支开发后, 需要做两件事:

- 进行代码的自我审查 (在3.2.1中描述)
- 将发生修改的文件添加到 commit list 中, 下述两种方法都可以实现这一点:
 - 通过 `git add your_filename` 指令, 将与 `your_filename` 对应的文件添加到 commit list 中
 - 通过 *sourcetree* 软件, 在 unstaged files 标签中, 选择文件将它添加到 staged files 中

3.1.3 提交代码(commit)

通过 `git commit` 指令提交代码

- 注意: `-m` 参数后需要添加一段提交注释

示例:

```
git commit -m "Modify CMakeList.txt to add a console when program running."
```

3.1.4 推送代码(rebase & push)

在推送代码之前, 建议将分支 `rebase` 在最新的 `dev` 分支上: 因为在 `checkout` 和 `push` 之间, 有可能别的开发者更新了 `dev` 分支:

```
git rebase dev
```

在成功rebase后, 再将当前分支推送到远程仓库中:

```
git push -u origin your_new_branch
```

3.1.5 命令合集

开发过程应***至少***使用如下指令:

```
git checkout -b your_new_branch
... // Development begins
...
... // Development ends
git add ...
git commit -m "Your meaningful commit message."
git push -u origin your_new_branch
```

完成推送代码后, 方可在 `gitlab` 上发起合并请求。

3.2 代码审查

3.2.1 自我审查

在两个环节应进行代码的自我审查:

- 使用 `git add` 指令, 提交代码时
- 使用 `git rebase` 指令时

审查的内容包括但不限于:

- 代码是否符合程序语言的最新标准 (e.g. ``ES6/7``, ``C++17``)
- 代码是否符合项目代码规范 (e.g. ``Google Code Style Guide``)
- 代码是否可以写得更简洁、可读、高效

3.2.2 他人审查

在向 **gitlab** 发起合并请求后, 应有其他项目成员进行代码审查, 并由该位成员负责合并。

4. 练习

4.1 联系 1: 在 **gitlab** 中创建一个团队项目

- 1) 创建一个小组(Group)
- 2) 创建一个属于这个小组的项目
- 3) 往这个小组中添加成员

4.2 练习 2: 分支管理

- 1) 基于 **master** 创建新的分支 **temp**

```
git checkout -b temp
```

- 2) 在 **temp** 下修改代码, 并将代码推送到远程

```
git commit -m "Your message."  
git push -u origin temp
```

在 **SourceTree** 中观察当前的分支状况 (图形显示)

- 3) 基于 **master** 创建新的分支 **dev**

```
git checkout master  
git checkout -b dev
```

- 4) 在 **dev** 下修改代码, 并 **commit**

```
git commit -m "Your message."
```

在 **SourceTree** 中观察当前的分支状况 (图形显示)

- 5) 删除远程的分支 **temp**

```
git push -u origin :temp
```

- 6) 删除本地的分支 **temp**

```
git branch -D temp
```

4.3 练习 3: 通过 `git rebase -i` 将多个 **commit points** 压缩成一个

- 1) 基于 **dev** 创建新的分支 **A**
- 2) 基于 **dev** 进行 3 次的 **commit**, **commit** 消息分别为

```
- Temp commit1
```

- Temp commit2
- Temp commit3

3) 使用 git rebase 压缩 commit points

```
git rebase -i HEAD~3
```

通过上述命令, 请仔细阅读你将在 `vi` 或 `vim` 中看到的如下文本:

```
pick bd99d06 Temp commit1
pick 09a34e5 Temp commit2
pick afac113 Temp commit3

# Rebase d2d5718..afac113 onto d2d5718 (3 command(s))
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
```

前三行的顺序是可以任意替换的, 这将导致最终呈现的 commit 顺序不同。

在此, 我们要做的是压缩 commit2 和 commit3, 所以我们只需要将对应的 `pick` 改成 `squash` 或 `s`, 再保存退出 (`:wq`) 即可。

4.4 练习 3: 生成一份关于最新的 commit 的 patch

- 1) 基于 dev 创建新的分支 A
- 2) commit 一次修改

```
`git commit -m "My commit"`
```

3) **自己动手**, 生成一份关于 `My commit` 的一份 patch

5. 总结

本文先给出了代码***管理工具***的下载地址, 描述了项目的***创建方式***, 再以***分支管理***、***代码审查*** (自我审查与他人审查) 为例, 指出了在开发过程中应遵循的行为规范和实现方式。最后, 提供了相关练习。

希望大家在今后的开发过程中, 能遵守规范, 掌握在*命令行*或图形界面下进行代码管理的方法。